UCRL-CONF-208613

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

A New Vista in Scientific Data Management

J. Keasler

December 15, 2004

Nuclear Explosives Code Developers' Conference
Livermore, CA, United States
October 4, 2004 through October 7, 2004

A New Vista in Scientific Data Management

Jeff Keasler

Lawrence Livermore National Laboratory, Livermore, California, 94550

This paper examines the use of an in-core database in place of traditional data structures in two ASC codes. Results so far indicate a dramatic reduction in code, coupled with an increase in functionality. Performance impact has been around one percent for the majority of problems tested. Debugging has been the major roadblock thus far, and a portable browser tool has been written that will soon work with a variety of debuggers.

Introduction

Within the last ten years, many 2D physics codes have been rewritten to work in 3D geometries. In many cases, the rewrite has offered little in the way of new physics capability, yet an enormous effort has been spent on "computer science" issues. In examining the old and new codes, there is often not a fundamental improvement in software methodology. If "computer science" issues are costing us so much time in development and getting in the way of our physics work, how can we say fundamental improvements in software design are not as important to address as the physics? This paper examines how the use of database technology can help increase the ratio of physics to non-physics programming done during the code development process.

The idea of using an in-core data registry (called a *database* in this paper) in scientific software is not new. However, many data management systems in the literature tend to be ad hoc, hard to learn due to the size of their API, hard to use due to their conceptual complexity, or they tend to have low performance. In other cases, the overall functionality provided by these systems is good and improving, but there are still barriers to widespread usage.

The database described in this paper is called Vista. Vista is simple to learn, relatively efficient, and easy to implement as a small library.

The purpose of this paper is not to convince others to use Vista, but rather to get people to examine their own software and evaluate the pros and cons of using a database in place of traditional data structures. While a database will not be appropriate for all applications, it is appropriate for many science based codes, and its benefits are worth exploring.

The Vista Database

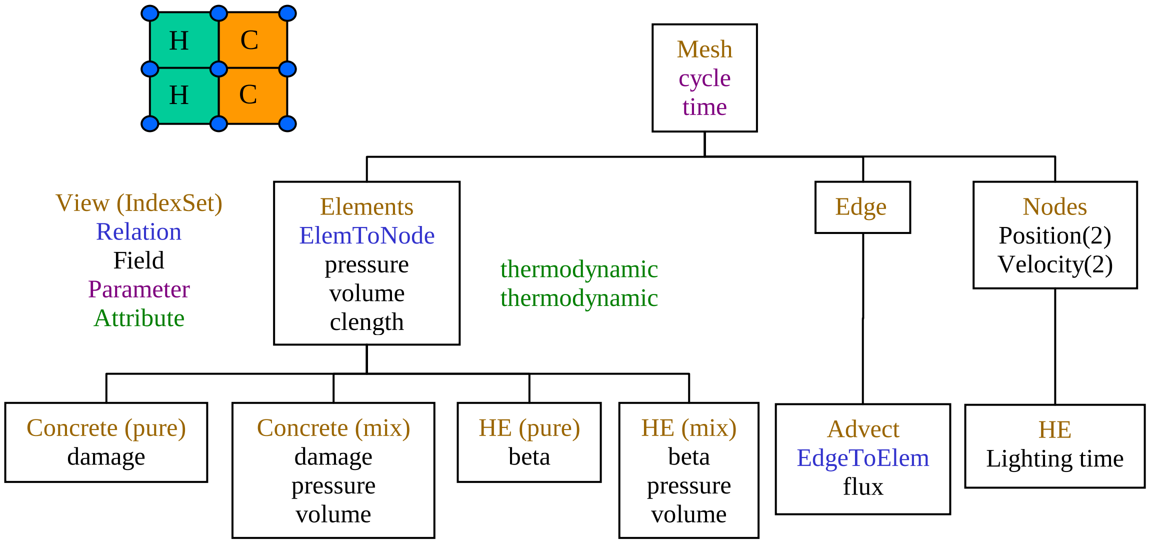


Figure 1. Database diagram for the four finite element, two dimensional mesh shown. The legend below the mesh highlights Vista's key features. Each box in the data diagram represents a View.

At the core of Vista is a data structure called a View. A View encapsulates a subset of a mesh (Fig. 1). Views can be arranged in a scoped hierarchy to efficiently encapsulate data. A View is defined to be an IndexSet and four Tables:

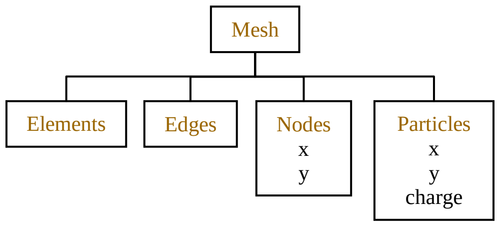
- IndexSet – Contains identifiers for all mesh entities contained in the View.
- Relation Table – Contains topological relationships between this View and other Views.
- Field Table – Contains discrete Fields which hold the bulk data for the View.
- Parameter Table – Contains Parameters which hold the non-bulk View data.
- View Table – Contains child Views.

A Table is a container such as an array or a hash table. A Vista Table is an extension of a traditional key/value table in that each table entry has additional properties:

- Aliasing – The ability to use two or more keys to refer to the same value.
- Attributes – The ability to attach additional information to each table entry.
- Depth – The ability to manage multiple copies of each table entry.

Vista's simple definition allows for a small and highly efficient implementation. Its consistent structure allows the user to replace many brute-force or cut-paste-modify sections of code with a compact, consolidated algorithm. Vista is not appropriate as a replacement for all data structures, but is powerful and elegant where it can be leveraged.

C++ implementation of Vista



```
View *mesh = new View(“mesh”);
View *particles =
   mesh->viewCreate(“particles”,num);
particles->fieldCreateReal(“x”);
particles->fieldCreateReal(“y”);
particles->fieldCreateReal(“charge”);

double *x = particles->fieldReal(“x”);
double *y = particles->fieldReal(“y”);
```

Figure 2. This example demonstrates usage of a C++ implementation of the Vista database.

Although the Vista specification is language neutral, C++ was chosen for the reference implementation. Only the basic encapsulation features of C++ were used, and care was taken to avoid constructs that could introduce portability or performance problems.

A C++ code sample is shown in Fig. 2. This code sample creates “Mesh” and “Particles” Views. The code required for the creation of the other Views is similar.

Note that character strings are used in the code to identify database variables. Integer tags could be used just as easily, and can be implemented much more efficiently.

In the code, the x coordinate field is retrieved using `particle->fieldReal(“x”)’. This syntax is similar to a C language struct member reference, `particle->x’. There are extra characters to type when using Vista, but the benefits described below far outweigh this cost.

Vista in ALE3D

ALE3D is a large parallel hydrodynamics code developed at Lawrence Livermore National Laboratory. Vista is currently being used to replace the high-level mesh objects in ALE3D. Thus far, all of the node centered data used in ALE3D has been moved to Vista. The node centered data is fully parallel, and exists over many subsets. This data represents about a third of all field data, and was previously stored in C language structs.

ALE3D has seen a negligible performance impact for moving to Vista. The performance impact is small because ALE3D can amortize each database lookup over a vector of calculations. We have tested the limits of this by operating on vectors as small as 64 values. In this case, we saw about a fifteen percent performance impact, however this is good considering the database can still be optimized.

Converting ALE3D to Vista has required a large amount of work. The greatest effort has been spent on maintaining backward compatibility with existing data structures as we transition to the new. The reference implementation of Vista offers some help here by allowing users to register pointers to exiting data structures in the database. This registration capability also makes it fairly easy to support legacy physics packages.

Restart Files

```
TransferView(Xfer *xfer, View *view) {

   xfer->enterScope(view->name) ;

   TransferIndexSet  (xfer, view->indexSet()) ;   // Transfer each
   TransferFields    (xfer, view->fields()) ;     // major part of
   TransferRelations (xfer, view->relations()) ;  // the current
   TransferParameters(xfer, view->parameters()) ; // View.

   // Transfer all the children
   for (View::Iter child(view->views()); !child.end(); child.next())
      TransferView(xfer, child.item()) ;

   xfer->exitScope(“..”) ;
}
```

Figure 3. This figure contains 14 of the 218 lines of driver code needed to write a restart file.

The Vista algorithm used to dump a restart file demonstrates the dramatic advantage to be gained by using a database in place of traditional data structures. Fig. 3 shows how a recursive procedure can be used to write a View tree.

The View tree contains many, but not all, of the maps, fields, parameters, and attributes used by ALE3D. The View tree as represented in the restart file is identical to the View tree representation in memory which is again identical to the View tree used in whiteboarding. The use of a database produces end-to-end consistency in the data model which helps to simplify code development and analysis.

The TransferView() routine in Fig. 3 accepts a data transfer class (Xfer) as an argument. The transfer class is an interface to low level database file formats. The data Xfer operator for writing is defined by five methods/functions:

- enterScope() – used to move into a new level of scope.
- arrayReal() – used to transfer an array of 64 bit real valued data.
- arrayInt() – used to transfer an array of integer data.
- string() – used to transfer an array of character data.
- exitScope() – used to move out of a level of scoping.

The simplicity of this interface makes it highly portable. It also allows ALE3D to fully leverage tools that have been written for multiple database file formats.

ALE3D currently supports four file formats: PDB, HDF5, XML, and XDMF. The transfer class for each of these requires about 120 text lines, and each supports extensive error checking and the ability to handle strided data.

Overall, it takes around 300 lines of code to write a complete restart file using Vista. This is a reasonably good improvement over the O(5000) lines of code used by many large science codes.

Network I/O

ALE3D is exploring the use of Vista in place of its current communication data structures. This is cutting edge development and results look promising. We have replaced a file containing 1500 lines of code with around 400 lines of code.

In the original algorithm, we allocated a communication buffer then packed fields member by member into the buffer. Next, we communicated the buffer and unpacked fields member by member. We did this for fifteen different communication modes, each acting on different fields which were defined over different mesh subsets. Since the mesh subsets were not in any way associated with the field data, there was a large amount of brute force code to grab the appropriate subset and field on a field by field basis as we did the packing and unpacking.

In our new algorithm using the database, we can use attributes to mark fields that need to be communicated. When we enter the communication routine, we pass in an attribute name for the communication we want to perform and query the database for all fields having that attribute. Attribute queries have been optimized in the database to return a vector of fields. We loop over the vector of fields, pack the data, communicate the data, and unpack the data into a vector of receive fields. Each database field is implicitly associated with a subset, so this algorithm is compact and almost trivial to implement.

The new algorithm results in much cleaner and compact code. ALE3D currently has over ten source code files for communicating data between domains. These files are used for communicating element centered data, material based data, mixed material data, etc. Each of these files is loosely based on the same template found in the first file we converted. If all goes well, we should be able to replace all of these files with a single file containing the new methodology. This should greatly reduce code maintenance costs.

Future Work

We have demonstrated in this section how Vista works for file and network I/O. The techniques described above should also be applicable to slide surfaces, user defined “edit” variables, fine grained dynamic load balancing, adaptive mesh refinement, various integration point schemes, and iterative convergence schemes.

Having a database will also allow us to easily reorganize our field data for cache efficiency. Mowry [1] found that by merely changing the memory layout used in the VPENTA benchmark from the SPLASH suite, a factor of three increase in performance was achieved. Likewise, Edwards [2] has at times seen noticeable performance improvement by rearranging field data contained in the SIERRA data registry. While there is no guarantee similar modifications will increase performance for ALE3D, it’s something we were not able to even think about before our data was stored in a database.

Vista in Diablo

Diablo is a parallel Fortran90 multi-mechanics code developed at Lawrence Livermore National Laboratory. Vista currently handles Diablo’s restart capability. Vista was chosen because it supports HDF5 and offers a pathway for using the VisIt visualization tool, which will be discussed below.

An added bonus of using Vista is its ability to write restart files in a form closely mirroring Diablo’s internal data structures and its ability to support new database file formats with a very small amount of code.

Vista supports Diablo through a Fortran interface. The interface assists in reading data from a restart file and/or registering pointers to Diablo’s data structures. The interface also supports the ALE3D restart file write routine described in this paper.

Overall, the use of Vista in Diablo has been a success, but there have been three awkward areas, mostly arising from the Fortran interface:

- Memory allocation – Fortran90 memory allocation is done by Diablo, so Vista has to point to Fortran90 data. This removes some of Vista’s flexibility.
- Array swapping – The Vista database has an intrinsic array swapping capability through the “depth” mechanism described above. It is awkward to access this functionality through the Fortran interface. This is one area that can be improved in the future.
- Reading vs. initializing data is conceptually awkward – The routine that initializes the Vista database to point at Diablo data structures is only called once during a code run. If the routine is called at timestep zero, it is only responsible for allocating memory. If the routine is called to read a restart file, then it must query data from the restart file and allocate memory. This dual use of code has dramatically shrunk the size of the restart code when compared to other file formats, but on the other hand it has also increased conceptual complexity. We are currently looking into ways to simplify this code model.

While Vista could benefit by improving its Fortran interface, it nonetheless has a fairly good mechanism for interacting with legacy code.

Visualization

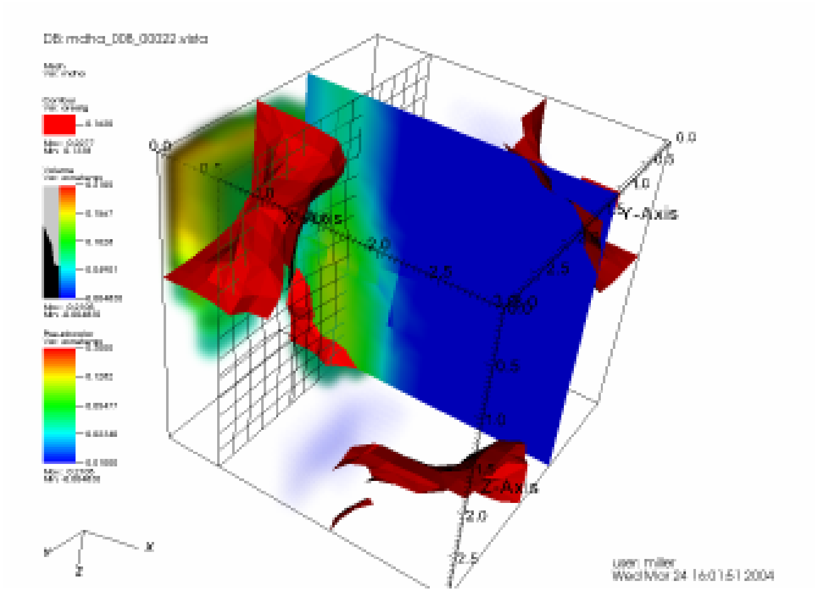


Figure 4. Selected fields of a Vista restart file as plotted by the VisIt visualization tool.

Vista restart files can be viewed with the VisIt tool developed by Lawrence Livermore National Laboratory (Fig. 4), or by the ParaView tool developed by Kitware.

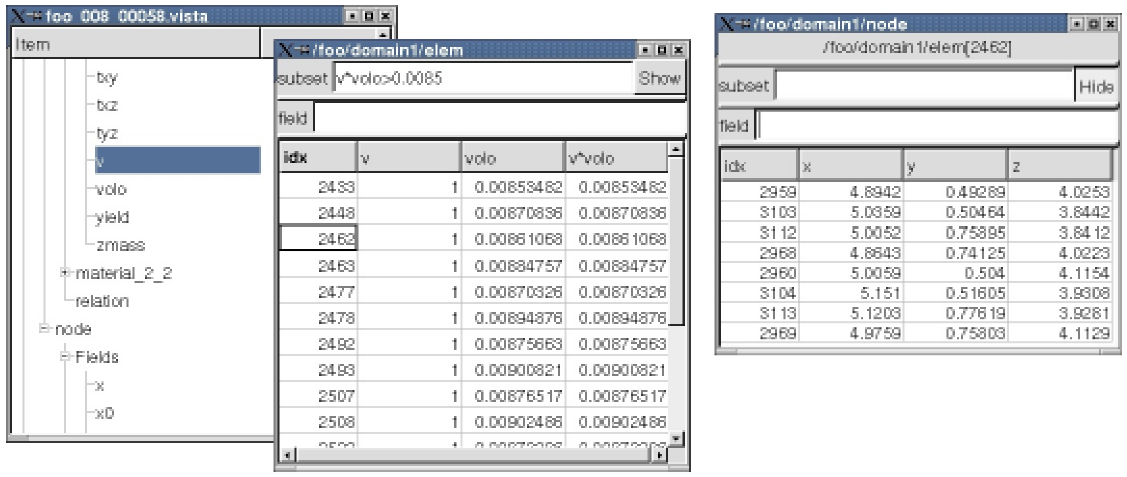
Currently, VisIt supports the Vista database through a specialized plug-in that works with single part or multi-part Vista restart files. The plug-in can display 2D quadrilateral or 3D hexahedral meshes produced by ALE3D's 2D and 3D run modes. The plug-in can also display the 3D meshes produced by Diablo.

Although the current VisIt plug-in is specialized for ALE3D and Diablo restart files, there is enough topological information embedded within Vista restart files to write a generalized Vista plug-in[1]. A generalized plug-in would decrease or even eliminate the need for interactions between code teams and visualization teams, even when the code team wants to change the form or contents of their database. It would also open the door for code teams to be able to visualize arbitrary subsets of their data, rather than only the restrictive subset of data that a particular file format such as SILO will support.

ParaView is a public domain visualization tool supported by several national laboratories. Vista in ALE3D can write the XDMF format supported by ParaView. XDMF is a combination of HDF5 and XML. Due to Vista's simplicity, it took about two hours to write a driver to output ALE3D data for this format.

[1] A small amount of additional meta-data must be specified as attributes on Views in the database.

Debugging



subset: v*volo>0.0085

idx	v	volo	v*volo
2433	1	0.00853482	0.00853482
2448	1	0.00870836	0.00870836
2462	1	0.00861068	0.00861068
2463	1	0.00884757	0.00884757
2477	1	0.00870326	0.00870326
2478	1	0.00894876	0.00894876
2492	1	0.00875663	0.00875663
2493	1	0.00900821	0.00900821
2507	1	0.00876517	0.00876517
2508	1	0.00902486	0.00902486

idx	x	y	z
2959	4.8942	0.49289	4.0253
3103	5.0359	0.50464	3.8442
3112	5.0052	0.75895	3.8412
2968	4.8643	0.74125	4.0223
2960	5.0059	0.504	4.1154
3104	5.151	0.51605	3.9308
3113	5.1203	0.77619	3.9281
2969	4.9759	0.75803	4.1129

Figure 5. The Vista data browser is interactive and aware of topological relations between entities.

One of the major drawbacks of using a database in place of traditional data structures is debugging. Traditional debuggers support static data structures such as arrays and C language structs. They do not contain built-in capabilities to browse user defined run-time dynamic data structures such as linked lists, hash tables, and of course, our database.

To solve this problem, we wrote a database browser tool. This tool was written using the open source Qt GUI product distributed by Trolltech. This allowed us to have a portable browser tool with minimal effort. At present, the tool can only be used to browse files, but will soon be extended to cooperate with a variety of debuggers.

There are three windows displayed in Fig. 5. The window on the left displays the overall contents of the database. The window's title bar is labeled with a mesh descriptor (the 58th time step of an eight domain problem). The user selects fields in this window for more detailed display.

The middle window in Fig. 5 is used to display fields in a database View. Here, the user has selected the *v* (relative volume) and *volo* (reference volume) fields which reside in the *domain1/elem* View (as labeled in the title bar of the window). Since both fields are in the same View, only one window is displayed. The user has also created a derived variable, *v*volo*, which gives the total volume for each element. A subset mask is applied showing only elements where the total volume is greater than 0.0085. Note that element 2462 is highlighted in the middle window, and the window on the right in Fig. 5 contains further information about this element.

The window on the right is the *domain1/node* View for element 2462. This window appears when an element is selected in the *domain1/elem* window because there is a topological relationship registered in the database associating each hexahedral element with the eight nodes that define it. Note that the nodal integer identifiers for these eight nodes are displayed in the window on the right. When that window initially pops up, only those ids are displayed. The user must specifically request which nodal fields they want to examine via the “field” form entry box in the window (here, x, y, z were selected).

Note that all windows are real-time interactive. The user can select a new element in the middle window to instantaneously update the details in the rightmost window.

No matter how clever the user, browsing this information in a debugger using traditional data structures takes a great effort. Using a browser tool that works in conjunction with a debugger, it takes just a few clicks of a mouse. Since debugging can involve a large amount of data browsing, it is easy to see how the use of a database and a browser tool should cut debugging time by an order of magnitude.

Conclusion

Realm vs. Purpose	**Isolation**	**Integration**	**Storage**
Physical	IndexSet	Relation	Field
Conceptual	View	Attribute	Parameter

Table 1 Vista contains all the features necessary for analysis and synthesis of data.

When compared to traditional data structures, the Vista database significantly simplifies portions of code that require a large amount of subset manipulation or data movement. There are two key reasons for this:

- Topological relationships between database entities are an integral part of the database specification. In fact, most of the key concepts in Vista map directly to subsetting capabilities (Table 1).
- All field data is stored in the database, rather than just a portion, making it possible to apply general transformations to data with fairly small algorithms.

As this paper has shown, using a database has many advantages. The consistent data layout provided by a database allows the programmer to write generic algorithms in a few hundred lines of code that can replace thousands of traditional lines of code. The look-and-feel of the database in the source code is not very different from the look-and-feel of C language structs, making the database easy to learn and use by non computer scientists. There is an end-to-end consistency in the data model across the white board, the internal data structures, the restart files, the visualization tool, and the debugger tool. Debugging may be simplified by an order of magnitude due to enhanced interactivity. Complexity is reduced because there are fewer lines of code, fewer data structures to remember, and fewer data models to remember. There also tends to be increased flexibility in data intensive algorithms, which are typical of computer science centric code.

Acknowledgements

The work presented in this paper is a formal organization of code work done by a large group of people. I would like to thank Edward Luke, Albert Nichols III, Rob Neely, Ivan Otero, Dan Malmer, Mark Miller, Shawn Dawson and many others for the countless hours of discussion, real world work, and experimentation that lead to the definition and implementation of Vista.

This work was performed under the auspices of the U.S. Dept. of Energy by the University of California, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94551 under contract No. W-7405-Eng-48.

References

Mowry, Todd, "Tolerating Latency Through Software-Controlled Data Prefetching," PhD Dissertation, Stanford University, March 1994.

Edwards, H. Carter, Sandia National Laboratory, Albuquerque, New Mexico, private communication (2004).

Edwards, H. Carter, "SIERRA Framework Version 3: Core Services Theory and Design," Sandia National Laboratory, Albuquerque, New Mexico, SAND REPORT SAND2002-3616.

Luke, Edward, "A Rule-Based Specification System for Computational Fluid Dynamics," PhD Dissertation, Mississippi State University, December 1999.

Gerlach, Jens, "Domain Engineering and Generic Programming for Parallel Scientific Computing", PhD Dissertation, TU Berlin, July 2002.